Aggregation Transformation of XML Schemas to Object-Relational Databases

Nathalia Devina Widjaya[1], David Taniar[1], and J. Wenny Rahayu[2]

[1] Monash University, School of Business Systems, Vic 3800, Australia
{Nathalia.Widjaya, David.Taniar}@infotech.monash.edu.au
[2] La Trobe University, Department of Computer Science and Engineering,
wenny@cs.latrobe.edu.au

Abstract. As XML has become an emerging standard for information exchange on the World Wide Web, it has gained attention in database communities to extract information from XML sees as a database model. Recently, many researchers have addressed mappings from XML structures onto database structures. In this paper, we present the way to transform the XML encoded format, which can be treated as a logical model, to the ORDB format. Firstly, the paper discusses the modelling of XML and why we need the transformation. Then, a number of transformation steps from the XML schema to the ORDB, with the emphasis on the transformations of aggregation relationships are presented. Two perspectives regarding this conceptual relationship (existence dependent aggregation which consists of homogeneous and ordered composition and independent aggregation) and their transformations are mainly discussed.

1. Introduction

The popularity of XML (eXtensible Markup Language) is growing and XML schema is being widely used to describe data. XML has emerged and is gradually accepted as the standard for describing data and interchanging data between various systems and databases on the Internet [1]. At the moment, XML offers the Document Type Definition (DTD) as formalism for defining the syntax and structure of XML documents. Then XML Schema definition language as a substitution of DTD provides more rich facilities for defining and constraining the content of XML documents [10].

With the wide acceptance of the Object Oriented conceptual models, more and more systems are initially modeled and being expressed with OO notation. This situation suggests the necessity to integrate the OO conceptual models and XML. The used of XML and XML Schemas to restore the data is no longer effective and efficient to store a lot of data. Because of that, there is a need to put the data from XML into the database without eliminating the object-oriented features that exist in XML Schemas.

The goal of this work is to present a coherent way to transform the XML schema into ORDB (Object-Relational Databases) using Oracle 9i features models. The

T. Böhme, G. Heyer, H. Unger (Eds.): IICS 2003, LNCS 2877, pp. 251-262, 2003.

emphasis of this paper is only on the transformation of aggregation relationship from XML schema in order to help people conveniently and automatically generate Oracle database. This transformation is important so that all the tables that are created using XML schema can be transformed to the object relational databases using Oracle format and features.

The work presented in this paper is actually part of a larger research project on Transformation from XML Schema to Object-Relational Databases. This project consists of three stages: (i) transformation association relationship from XML Schema to Object-Relational Database, (ii) transformation inheritance relationship from XML Schema to Object Relational Database and (iii) transformation aggregation relationship from XML Schema to Object Relational Database. The research results from the first and second stage have been reported in [8] and [9]. In this paper, we focus on the final stage of the project.

The content of the article will consist of the introduction about XML schema and ORDB. We will explain the relationship that can exist in the XML schema and OO (Object-Oriented) concepts. The remainder of the paper is organised as follows. Section 2 discusses about the overview over OO concepts and OO in XML schemas. Then, we review some closely related work. Section 3 presents several generic-transforming rules from XML schema to ORDB with the emphasis on the transformation of aggregation relationship. We discuss the transformation steps and give example for each of them. Section 4 concludes the paper and further work that can be done.

2. Background and Related Work

2.1 Object-Oriented: A Brief Overview

In 1970 there was only Relational Database Management System (RDBMS) [2]. Traditional RDBMSs perform well only when working on numeric data and characters stored in tables, what are often called "simple data types." [8] Then, ORDBMS (Object-Relational Database Management System) comes later to improve RDBMSs performance. Basically, the ORDBMS is the RDBMS with the object-oriented features. ORDBMS becomes popular because of the failure of ODBMSs, which has limitations that can prevent it from taking on enterprise-wide tasks. Therefore, by storing objects in the object side of the ORDBMS but keeping the simpler data in the relational side, users may approach the best of both worlds. For the foreseeable future, however, most businesses data will continue to be stored in object relational database system.

Since ORDBMS has object-oriented features, we will discuss briefly about Object-Oriented Conceptual Model (OOCM). OOCM encapsulates the structural/static as well as behavioural/dynamic aspects of objects. The static aspects consist of the classes/objects and the relationship between them, namely inheritance, association and aggregation. The dynamic aspect of the OOCM consists of generic methods and user-defined methods. We only discuss about the static aspects since this is the topic that is relevant with this paper. Static aspects in OOCM create objects and classes that also include decisions regarding their attributes. Furthermore, they

also concern on the relationship between objects. The basic segment of the object-oriented system is an object. An object is a data abstraction that is defined by an object name as a unique identifier, valued attributes (instance variables), which give a state to the object, and methods, or routines that access the state of the object.

In XML Schemas, there are static aspects from object-oriented conceptual model that we can find. The aggregation is one of OOCM features that we will discuss in this paper. Aggregation is a "part-of" relationship (refer to figure 1), in which a composite object ("whole") consists of other component objects ("parts").

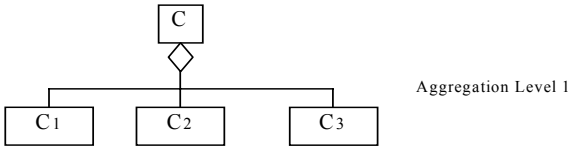

Figure 1. A one-levelled aggregation relationship rooted at *C*

There are four types of aggregation relationship according to Dillon and Tan (1993) such as: sharable dependent, sharable independent, non-sharable dependent and non-sharable independent composition. In this paper, we only focus on existence dependent and existence independent composition. It is vital to remember that in UML the term 'composition' refers to exclusive and dependent aggregation. However, we use composition interchangeably with aggregation and use qualifications to distinguish between the different categories. Furthermore, we also look at two more types of aggregation relationship, i.e. ordered composition and homogenous/heterogeneous composition.

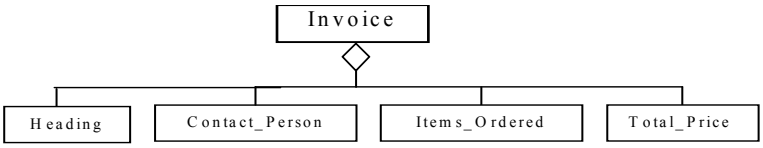

Figure 2. Class diagram showing existence dependent composition

Existence-dependent aggregation means there is a dependency between the "whole" object and its "part" object. In the existence-dependent, the deletion of the "whole" object will cause the deletion of that object and its elements (Refer to figure 2). While in existence-independent aggregation, there is no dependency between the "whole" object and its "part" object, therefore the deletion of the "whole" object will not cause the deletion of its element (Refer to figure 3).

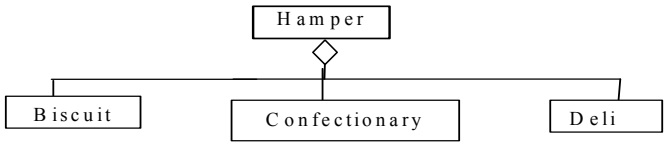

Figure 3. An existence independent composition example

We called the aggregation as ordered composition if a “whole” object composed of different “part” objects in particular order. In other words, the order of occurrence of the “part” objects in the composition is vital to the model.

The other types of composition are heterogeneous and homogeneous. Basically all categories of composition are heterogeneous since one “whole” object may consist of several different types of “part” objects. On the other hand, Homogenous composition means that one “whole” object consists of “part” objects that are of the same type (Refer to figure 4). The notation that is used to show the aggregation relationship is the diamond arrow. The class with the diamond next to it refers to the super class.

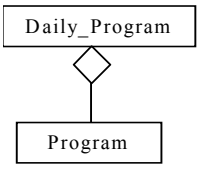

Figure 4. Class diagram showing homogeneous composition example

2.2 Related Work

Most existing work has focused on a methodology that has been designed to map a relational database to an XML database for database interoperability. The schema translation procedure is provided with an EER (Extended Entity Relationship) model mapped into XML schema [3].

There are many works that explain about the mapping from relational databases to XML. Some of them still use DTD [10,11] and some of them use XML schema [5]. Since XML is rapidly emerging as the dominant standard for exchanging data on the WWW, the previous work already discussed about mapping referential integrity constraints from Relational Database to XML, semantic data modeling using XML schemas and enhancing structural mapping for XML and ORDB.

In addition, the study about the use of new scalar and aggregate functions in SQL for constructing complex XML documents directly in the relational engine has been done [10].

Relational and object-relational database systems are a well-understood technique for managing and querying such large sets of structured data. In [5], the writers wrote about how a relevant subset of XML documents and their implied structure can be mapped onto database structures. They suggest mapping DTDs onto object-relational databases schemas and to overcome the typical problems (large schemas), they suggested an algorithm for determining an optimal hybrid database schema.

The way to model XML and to transform the OO conceptual models into XML Schema have been discussed in [10]. The writers choose the OO conceptual model because of its expressive power for developing a combined data model. They come out with several generic-transforming rules from the OO conceptual model to XML schema, with the emphasis on the transformations of generalization and aggregation relationships. The XML Schema code that is presented below, in this paper, is

adopted from the existing work that is done previously. In addition, our paper is done to improve what has been done in [10].

The work reported in this paper is distinguishes from this work in the following aspects. First, we focus the transformation from XML schema to ORDB. Second, our transformation target using OO features in Oracle 9i not just the general OO features. The similarity is we take aggregation relationships into consideration (existence dependent and independent aggregation).

3. Transformation from XML Schema to ORDB: The Proposed Methodology

In the following, we use XML Schema and Oracles 9i to interpret the aggregation relationship in OO conceptual models. We discuss the transformation or mapping from XML Schema to ORDB. In this section, we also validate the following documents against the schema. In addition, we also give the example how to insert the data to the table after creating the table in Oracle 9i. Table 1 shows the expressions that are used for data types from XML schema and ORDB in this article. Both of them have the same meaning, but using different phrase.

XML Schema Data types	ORDB Data Types
String	Varchar2
Decimal	Decimal(l,d); l = length, d = decimal

Table 1. Data Types Mapping

3.1 Existence Dependent (Ordered Composition)

The following steps generate a transformation from XML Schema to Object Oriented Relational Database in Oracles 9i for ordered existence dependent aggregation relationship.

(i) For an aggregation relationship rooted at a composite class C, an element named C with a complex Type Ctype in XML schema (<xsd:element name = "C" type= "Ctype">) can be transformed by creating a cluster named C_cluster in ORDB. Then, write the type of class C attributes (such as C_id). Usually it is in the *varchar2* format and the user will enter the length for it. Refer to table 1.0 to transform the data types from XML schema to ORDB.

XML Schema:

```
<xsd:element name="Invoice" type = "InvoiceType"/>
    <xsd:complexType name ="InvoiceType">
```

ORDB:

```
Create Cluster Invoice_Cluster
    (invoice_Id varchar2 (10))
```

(ii) Create a table for composite class *C* and the type of its attribute, which is exactly the same as `C_cluster` above and has *Not Null* besides the `C_id` which means table invoice must have an id. Then, create a primary key for this table, which is usually `C_id`. Next, create a cluster as *C* table attributes and the type will be `C_cluster (C_id)`.

ORDB:

```
Create Table Invoice
    (invoice_id        varchar2(10) Not Null,
     Primary Key       (invoice_id))
Cluster Invoice_Cluster (invoice_id);
```

(iii) Based on each sub-element named C1 within the *complexType Ctype* in the XML Schema (`<xsd:element name = "C1" type= "...">`), we need to create another table for each of sub-element. Its attributes will consist of `C_id`, `C1_id` and other attributes that are relevant with `C1`. `C_id` and `C1_id` will be the primary key and the foreign key will be *C_id references C (C_id)*. Next, create a cluster and its type that should be the same with the cluster that is created before, `C_cluster (C_id)`.

XML Schema:

```
<xsd:sequence>
    <xsd: element name = "Heading" type = "xsd:string"/>
```

ORDB:

```
Create Table Heading
    (invoice_id        varchar2 (10)  Not Null,
     heading_id        varchar2 (10)  Not Null,
     Headingvarchar2 (30),
     Primary Key (invoice_id, heading_id),
     Foreign Key (invoice_id) References Invoice (invoice_id))
Cluster Invoice_Cluster (invoice_id);
```

(iv) Create index for `C_cluster_index` on cluster `C_cluster`

ORDB:

```
Create Index Invoice_Cluster_Index
    On Cluster Invoice_Cluster;
```

Below is the full example of the transformation from ordered existence dependent aggregation relationship from XML Schema to ORDB.

XML Schema for ordered existence dependent aggregation relationship:

```
<xsd:element name= "Invoice" type = "InvoiceType"/>
<xsd:complexType name = "InvoiceType">
    <xsd:sequence>
        <xsd:element name = "Heading" type = "xsd:string"/>
        <xsd:element name = "Contact_Person" type =
        "ContactPersonType"/>
```

```
            <xsd:element name = ″Items_Ordered″ type = ″xsd:string″/>
            <xsd:element name = ″Total_Price″ type = ″xsd:decimal″>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name = ″ContactPersonType″>
        <xsd:sequence>
            <xsd:element name = ″Name″ type = ″xsd:string″/>
            <xsd:element name = ″Address″ type = ″xsd:string″/>
            <xsd:element name = ″PhoneNo″ type = ″xsd:decimal″/>
        </xsd:sequence>
    </xsd:complexType>
```

ORDB for ordered existence dependent aggregation relationship:

```
Create Cluster Invoice_Cluster
    (invoice_id         varchar2 (10));
Create Table Invoice
    (invoice_id         varchar2 (10) Not Null,
     Primary Key        (invoice_id))
     Cluster            invoice_cluster (invoice_id);
Create Table Heading
    (invoice_id         varchar2 (10) Not Null,
     heading_id         varchar2 (10) Not Null,
     Primary Key        (invoice_id, heading_id),
     Foreign Key        (invoice_id) References Invoice (invoice_id))
Cluster                 invoice_cluster (invoice_id);
Create Table Contact_Person
    (contact_person_id          varchar2 (10) Not Null,
     invoice_id         varchar2 (10) Not Null,
     name               varchar2 (40),
     address            varchar2 (40),
     phone_no           number
     Primary Key        (invoice_id, contact_person_id),
     Foreign Key        (invoice_id) References Invoice (invoice_id))
Cluster                 invoice_cluster (invoice_id);
Create Table Item_Ordered
    (invoice_id         varchar2 (10) Not Null,
     item_ordered_Id    varchar2 (10) Not Null,
     Primary Key        (invoice_id, item_ordered_id),
     Foreign Key        (invoice_id) References Invoice (invoice_id))
Cluster                 invoice_cluster (invoice_id);
Create Table Total_Price
    (invoice_id         varchar2 (10) Not Null,
     total_price_ID     varchar2 (10) Not Null,
     Primary Key        (invoice_id, total_price_id),
     Foreign Key        (invoice_id) References Invoice (invoice_id)
Cluster                 invoice_cluster (invoice_id)
Create Index Invoice_Cluster_Index On Cluster Invoice_Cluster
```

3.2 Existence Dependent (Homogeneous Composition)

Figure 3 shows the homogenous existence dependent aggregation relationship. We can generate a transformation for existence dependent homogeneous aggregation relationship from XML Schema to ORDB in Oracle 9i as follows.

(i) Each sub-element named `C1` with a complex Type `Ctype` in XML schema (`<xsd:element name = "C1" type= "…." MinOccurs= "…" maxOccurs= "…">`) need to be created as an object named `C1`. Then, write the type of `C1` attributes (such as `C1_id`), usually it is in the `varchar2` format, and the user will enter the length for it.

XML Schema:

```
<xsd:element name = "Program" type = "xsd:string"
    minOccurs="1" maxOccurs="unbounded"/>
```

The *maxOccurs* explains the maximum number of *Program* in *Daily_Program.* This may be a positive integer value or the word unbounded to specify there is no maximum number of occurrences. The *minOccurs* shows the minimum number of times an element may appear. It is always less than or equal to the default value of *maxOccurs*, i.e. it is 0 or 1. Similarly, if we only specify a value for the *maxOccurs* attribute, it must be greater than or equal to the default value of *minOccurs*, i.e. 1 or more.

ORDB:

```
Create Or Replace Type Program As Object
    (Program_id       varchar2 (10));
```

(ii) Create a table for composite class `C1` (as a table of the object above)

ORDB:

```
Create Or Replace Type Program_Table As Table Of Program
```

(iii) For a homogeneous existence dependent aggregation relationship rooted at a composite class `C`, an element named C within the `complexType Ctype` in the XML Schema (`<xsd:element name = "C" type= "Ctype">`) need to created as an object named C. Its attributes will consist of `C_id`, and other attributes that are relevant to it. `C_id` will be the primary key. Next, nested this table and stored it as the table that is created before.

XML Schema:

```
<xsd:element name = "Daily_Program" type = "DailyProgramType"/>
    <xsd:complexType name = "DailyProgramType">
```

ORDB:

```
Create Table Daily_Program
    (daily_program_idvarchar2(10) Not Null,
```

```
program_name            Program_Table,
Primary Key (daily_program_id))
Nested Table program_name Store As Program_Table;
```

Below is the full example of transformation from XML schema homogeneous existence dependent aggregation to ORDB.

XML Schema for homogeneous existence dependent aggregation

```
<xsd:element name; ″Daily_Program″ type = ″DailyProgramType″/>
<xsd:complexType name = ″DailyProgramType″>
    <xsd:element name = ″Program″ type = ″xsd:string″ minOccurs = ″1″
        maxOccurs= ″unbounded″/>
```

ORDB for homogeneous existence dependent aggregation

```
Create Or Replace Type Program As Object
    (program_id             varchar2 (10));
Create Or Replace Type Program_Table As Table Of Program
Create Table Daily_Program
    (daily_program_idvarchar2(10) Not Null,
     program_name           Program_Table,
     Primary Key (daily_program_id))
     Nested Table program_name Store As Program_Table;
```

3.3 Existence Independent

The following steps generate a transformation from XML Schema to Object Oriented Relational Database in Oracle 9i for existence independent aggregation relationship.

(i) For an aggregation relationship rooted at a composite class C, an element named C with a complex Type Ctype in XML Schema (<xsd:element name = “C” type = “Ctype”>) can be transformed by creating a table named C in ORDB. Then, write the type of class C attributes (such as C_id) based on the attribute for that Ctype in the XML schemas.

XML Schemas:

```
<xs:element name="Hamper" type ="HamperType">
……
<xs:element name = "HamperType">
<xs:complexType>
  <sequence>
  <xs:element name = "hamper_id" type = "xs:string"/>
    <xs:element name = "hamper_price" type ="xs:decimal"/>
    </sequence>
</xs:complexType>
</xs:element>
```

ORDB:

```
Create Table Hamper
(hamper_id          varchar2(3)     Not Null,
```

```
hamper_price        Number,
Primary Key (hamper_id));
```

ii) Create tables for each element under choice. The element reference under choice means that it refers to the details below where the element name equals to the element reference.

XML Schema:

```
<xs:complexType>
<xs:choice>
   <xs:element ref = "Biscuit"/>
   <xs:element ref = "Confectionary"/>
   <xs:element ref = "Deli"/>
</xs:choice>
</xs:complexType>

<xs:element name = "Biscuit">
<xs:complexType>
<sequence>
   <xs:element name = "biscuit_id" type =          "xs:string"/>
   <xs:element name = "biscuit_name" type ="xs:string"/>
   <xs:element name = "biscuit_price" type = "xs:decimal"/>

</sequence>
</xs:complexType>
</xs:element>
```

ORDB:

```
Create Table Biscuit
(biscuit_id          varchar2(3)      Not Null,
 biscuit_name        varchar2(20),
 biscuit_price Number,
 Primary Key (biscuit_id));
```

iii) Create the last table that we called as an aggregate table which will link the composite class with the sub-classes. Then, create the attributes for this class which includes the id for the composite class, part_id and part_type. Lastly, create a primary key and a foreign key.

ORDB:

```
Create Table Aggregate
(hamper_id           varchar2(3)      Not Null,
 part_id             varchar2(3)      Not Null,
 part_type           varchar2(20)     Check
(part_type In ('biscuit', 'confectionery', 'deli')),
 Primary Key (hamper_id, part_id),
 Foreign Key (hamper_id) References hamper (hamper_id));
```

Below is the mapping of ORDB for existence independent from the XML Schema existence independent aggregation.

ORDB for existence independent aggregation

```
Create Table Hamper
    (hamper_id      varchar2(3)     Not Null,
     hamper_price   Number,
     Primary Key (h_id));
Create Table Biscuit
    (biscuit_id     varchar2(3)     Not Null,
     biscuit_name   varchar2(20),
     biscuit_price  Number,
     Primary Key (biscuit_id));
Create Table Confectionery
    (confectionery_id           varchar2(3)
     confectionery_name         varchar2(20),
     confectionary_price        Number,
     Primary Key (confectionary_id));
Create Table Deli
    (deli_id        varchar2(3)     Not Null,
     deli_name      varchar2(20),
     deli_price     Number,
     Primary Key (deli_id));
Create Table Aggregate
    (hamper_id      varchar2(3)     Not Null,
     part_id        varchar2(3)     Not Null,
     part_type      varchar2(20)    Check
    (part_type In('biscuit', 'confectionery', 'deli')),
     Primary Key (hamper_id, part_id),
     Foreign Key (hamper_id) References hamper (hamper_id));
```

4. Conclusion and Future Work

In this paper, we have investigated the transformation from XML schema to the ORDB by using Oracle 9*i*. We emphasis the transformation of *aggregation relationship* to help people easily understand the basic object conceptual mapping that we proposed. This transformation is important because people always eliminate the object-oriented conceptual features when they transform XML schema to the database.

Our research gives better solution in transformation XML Schema into ORDB rather than the XML features that Oracle 9i have. Oracle 9i can only convert all the data or query result in XML format but it does not deal with the type of database that is used, such as relational database or object oriented database, like we do. This transformation can be applied on any XML documents that use XML Schema.

Our future work is being planned to investigate more transformation from XML schema to ORDB for other XML Schema features that has not been discussed in this paper. In addition, further research should be done to create a query from XML schema to get the data from the Oracle 9i databases.

References

1. Bray, T., Paoli, J., Sperberg-McQueen, C.M (eds)., "Extensible Markup Language (XML) 1.0. W3C (1998) http://www.w3c.org/TR/REC-xml
2. Dillon T and Tan PL, *Object Oriented Conceptual Models*. Prentice Hall, 1993.
3. Fong J., Pang F., and Bloor, C., " Converting Relational Database into XML Document". *Proc. 12th Intl. Workshop on Database and Expert Systems Appl*, 2001, pp. 61-65, 2001.
4. Klettke M. and Meyer H., " XML and Object Relational Database System". *Lecture Notes in Computer Science*, vol 1997, Springer-Verlag, pp.151-170, 2001.
5. Mani M., Lee D. and Muntz R., " Semantic Data Modelling Using XML Schemas". *Lecture Notes in Computer Science*, vol 2224, Springer-Verlag pp.149-163, 2001.
6. Shanmugasundaram, J. et al., "Efficiently publishing relational data as XML documents". *The VLDB Journal*, vol 10, pp. 133-154
7. Stonebraker, M., and Moore, D., "*Object-relational DBMSs:the next great wave*". Mogran Kaufmann Publishers. San Francisco, 1996.
8. Widjaya, N.D., Taniar, D**.**, Rahayu, J.W., and Pardede, E., "Association Relationship Transformation of XML Schemas to Object-Relational Databases", *Proceedings of the 4th International Conference on Information Integration and Web-based Applications and Services (IIWAS'2002*), pp. 135-142, 2002
9. Widjaya, N.D., Taniar, D., and Rahayu, J.W, "Inheritance Relationship Transformation of XML Schemas to Object-Relational Databases", *Proc. of the 4th Intl. Conference on Intelligent Data Engineering & Automated Learning (IDEAL'2003),* 2003.
10. Xiao R., Dillon T., Chang E., and Feng L., "Modelling and Transformation of Object-Oriented Conceptual Models into XML Schema". *Lecture Notes in Computer Science*, vol 2113, Springer-Verlag, pp.795-804, 2001.
11. Yang X. and Wang G., "Efficiently Mapping Referential Integrity Constraints from Relational Databases to XML. *LNCS*, vol 2151, Springer-Verlag, pp.338-351, 2001.
12. Yang X. and Wang G., "Mapping Referential Integrity Constraints from Relational Databases to XML. *LNCS*, vol 2118, Springer-Verlag, pp.329-340, 2001.